HY590.45
Modern Topics in Scalable Storage Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy590-45
Petal: Client view

- LFS (Petal Client)
- NT FS (Petal Client)
- PC FS (Petal Client)
- BSD FFS (Petal Client)

Scalable Network

A virtual disk

/dev/vdisk11  /dev/vdisk2  /dev/vdisk3  /dev/vdisk4  /dev/vdisk5
Goals of Petal

- Tolerate and recover from any component failure
- Geographically distribute to tolerate site failures
- Transparently reconfigure to expand, balance load
- Dynamically balance load
- Fast and efficient support for backups and recovery
Server modules

- Global State Module
- Liveness Module
- Recovery Module
- Data Access Module
- Virtual to Physical Translator
Virtual to physical mapping

\(<\text{vdiskID, offset}> \rightarrow <\text{serverID, diskID, diskOffset}>\)

- Server 0
  - VDir
  - GMap
  - PMap0

- Server 1
  - VDir
  - GMap
  - PMap1

- Server 2
  - VDir
  - GMap
  - PMap2

- Server 3
  - VDir
  - GMap
  - PMap3

- <\text{diskID, diskOffset}>
Support for backup

• Snapshots
  – Petal can quickly create an exact copy of a virtual disk at a specified point in time by using copy-on-write techniques
  – A snapshot is like any other vdisk, but cannot be modified
  – VDir: vdiskID → ⟨global-map-identifier, epoch-number⟩

• Crash-consistent snapshot
  – Similar to disk image left after an application crash
Reconfiguration

• Dynamic change in vdisk # of servers, redundancy

• How is it performed
  – Create new GMap with desired redundancy, server mapping
  – Change VDir entries that refer to old GMap to new one
  – Redistribute data to the servers according to new GMap, requiring substantial amounts of network and disk traffic
  – Read requests will be tried on new GMap first, then the old GMap if the translation has not yet been transferred
  – Writes are always performed on the new GMap

• Improve efficiency via fencing
Chained de-clustering

Virtual Disk

<table>
<thead>
<tr>
<th>Server 0</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
</tr>
<tr>
<td>D3</td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
</tr>
<tr>
<td>D7</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
</tr>
</tbody>
</table>
Data access and recovery

- How are reads performed
- How are writes performed
Data access and recovery

Read protocol
1. Try primary; if live, get read lock
2. If down, try secondary; if live, get read lock
3. Return block contents, release read lock

Write protocol
1. Contact primary
2. If alive, mark block busy in stable storage
3. Primary apply request locally and simultaneously send write to secondary
4. When both complete, clear busy bit, respond to client
5. If primary crashes during request, busy bit is used to recover later on
6. If primary dead, start from secondary
7. (Secondary checks primary indeed crashed)

If primary or secondary is down, on live node:
1. Mark data as *stale* before writing to disk
2. During recovery, make replicas consistent by exchanging *dirty-region log*
a read will see a state at least as recent as that produced by the most recently completed write that completed before the read started
Failure-free operation

if some reader sees the results of a particular write, then any reader that starts after that reader finishes will also see a result at least that recent.
Recovery from primary crash

write(b,6)

write

busy

reconcile

clear busy bit

ack

check status

restart

write

write

busy

ack
Recovery from primary crash

**Primary**
- Write (b,5)
- Busy
- Crash
- Restart
- Get dirty-region log
- Reconcile
- Clear busy bit
- Ack

**Secondary**
- Write (b,6)
- Busy
- Primary dead??
- Set stale
- Write
- Ack

**Replay Write**
Recovery from secondary crash

write(b,5)

write(b,6)

write

backup dead??

set stale

write

clear busy bit

ack

clear busy bit

write

ack

backup dead??

crash

write

restart

write

crash

get dirty-region log

reconcile

replay write

write

write

write

ack

ack
Petal cannot handle partitions

write(b,5)
Prototype

Diagram:

- Petal Client
- Petal Client
- Petal Client
- Petal Client

- Digital ATM Network

- Petal Virtual Disk

- Petal Server
- Petal Server
- Petal Server
- Petal Server

- Logs
Latency, throughput results

<table>
<thead>
<tr>
<th>Request</th>
<th>Local Disk</th>
<th>Petal RZ29 Log</th>
<th>Petal NVRAM Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 byte Read</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8 Kbyte Read</td>
<td>11</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>64 Kbyte Read</td>
<td>21</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>512 byte Write</td>
<td>10</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>8 Kbyte Write</td>
<td>12</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>64 Kbyte Write</td>
<td>20</td>
<td>40</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 1: Latency of a Chained-Declustered Virtual Disk

<table>
<thead>
<tr>
<th>Request</th>
<th>Normal</th>
<th>Failed</th>
<th>% of Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 byte Read</td>
<td>3150 req/s</td>
<td>2310 req/s</td>
<td>73%</td>
</tr>
<tr>
<td>8 Kbyte Read</td>
<td>20 Mbytes/s</td>
<td>14.6 Mbytes/s</td>
<td>73%</td>
</tr>
<tr>
<td>64 Kbyte Read</td>
<td>43.1 Mbytes/s</td>
<td>33.7 Mbytes/s</td>
<td>78%</td>
</tr>
<tr>
<td>512 byte Write</td>
<td>1030 req/s</td>
<td>1055 req/s</td>
<td>102%</td>
</tr>
<tr>
<td>8 Kbyte Write</td>
<td>6.6 Mbytes/s</td>
<td>6.6 Mbytes/s</td>
<td>100%</td>
</tr>
<tr>
<td>64 Kbyte Write</td>
<td>12.3 Mbytes/s</td>
<td>12.5 Mbytes/s</td>
<td>101%</td>
</tr>
</tbody>
</table>

Table 2: Normal and Failed Throughput of a Chained-Declustered Virtual Disk
Scaling

![Scaling Graph]

- 512 byte Read
- 8 Kbyte Read
- 64 Kbyte Read
- 512 byte Write
- 8 Kbyte Write
- 64 Kbyte Write

Relative Throughput vs Number of Servers